Chapter 4

INTERRUPTS ON THE TMS320LF2407

4.1 Introduction to Interrupts

The interrupts on the LF2407 allow the device hardware to trigger the CPU of the LF2407 (CPU=C2xx DSP core) to break from the current task, branch to a new section of code and start a new task, then return back to the initial task. The "new task" referred to in the previous sentence is known as the Interrupt Service Routine (ISR). The ISR is simply a separate user-written subroutine, which the core will branch to every time a certain interrupt occurs.

For example, say the ADC is being used and we want the program to load the conversion value into the accumulator every time the ADC finishes a conversion. The ADC can be configured to generate an interrupt whenever a conversion is finished. When the ADC generates its interrupt, the interrupt signal makes its way through the interrupt hierarchy to the core and the core then branches to the appropriate ISR.

In a more general sense, when an interrupt occurs, the core branches to the ISR (GISR1, GISR2 etc… depending on the interrupt) where an interrupt service routine is located. In the ISR, after the instructions are executed, the interrupt hierarchy is "reset" to allow for future interrupts. This usually entails clearing the peripheral level interrupt flag bit and clearing the INTM bit. These steps ensure that future interrupts of the same origin will be able to pass through to the core. The final instruction in the ISR is the RET command, which instructs the core to return to where it was before the interrupt occurred.

4.2 Interrupt Hierarchy

This section will explain the different hierarchical levels and how an interrupt request signal propagates through them. The different control registers and their operations will be reviewed.

4.2.1 Interrupt Request Sequence

There are two levels of interrupt hierarchy in the LF2407 as seen in Fig. 4.1 below. There is an interrupt flag bit and an interrupt enable bit located in each peripheral configuration register for each event that can generate an interrupt. The peripheral interrupt flag bit is the first bit to be set when an interrupt generating event occurs. The interrupt enable bit acts as a "gate". If the interrupt enable bit is not set, then the setting of the peripheral flag bit will not be able to generate an interrupt signal. If the enable bit is set, then the peripheral flag bit will generate an interrupt signal. That interrupt signal will then leave the peripheral level and go to the next hierarchal level.

Once an interrupt signal leaves the peripheral level, it is then multiplexed through the Peripheral Interrupt Expansion (PIE) module. The PIE module takes the many individual interrupts and groups them into six priority levels (INT1 through INT6). Once an interrupt reaches the PIE, a code identifying the individual interrupt is loaded into the Peripheral Interrupt Vector Register (PIVR). This allows the ISR to determine which interrupt was actually asserted when multiple interrupts from the same level occur. After passing through the PIE module, the interrupt request signal has now entered the upper level of hierarchy or the "CPU level".

The six interrupt groupings from the PIE module feed into the CPU level. The final stage of the CPU level is the CPU itself (C2xx core). From Fig. 4.1, we can see the six interrupt levels and the many individual peripheral interrupts assigned to priority level. Each of the six levels has a corresponding flag bit in the Interrupt Flag Register (IFR). Additionally there is an Interrupt Mask Register (IMR) which acts similar to the interrupt enable bits at the peripheral level. Each of the six bits in the IMR behaves as a "gate" to each of the corresponding six bits in the IFR. If the corresponding bits in both the IFR and IMR are both set, then the interrupt request signal can continue through to the C2xx core itself.

Once the interrupt request signal has entered the CPU level and has passed through the IFR/IMR, there is one more gateway the signal must pass through in order to cause the core to service the interrupt. The Interrupt Mask (INTM) bit must be cleared for the interrupt signal to reach the core. When the core acknowledges a pending interrupt, the INTM bit is automatically set, thereby not allowing any more interrupts from reaching the core while a current interrupt is being serviced.

When the core is finished with the current interrupt, only the flag bit in the IFR is cleared automatically. The INTM bit and the peripheral level flag bit must be cleared "manually" via software. When this is done, the core will acknowledge the highest priority pending interrupt request signal.

Additionally, if an interrupt request signal occurs, but the signal never reaches the core, all flag bits "downstream" of the point where the signal was halted will still remain set until cleared by software. The IFR bits will be cleared if: (1) the interrupt path to the core is opened, and the interrupt is acknowledged normally or (2) the bit is cleared "manually" by software. If no interrupt request has occurred but the peripheral level IF bit is set and the peripheral IE bit is later set without clearing the IF bit, then an interrupt request signal will be asserted and the corresponding IFR bit will be set.

Furthermore, in the event that two interrupts of different priority groupings (INTx) occur at the same time, the highest priority interrupt will be acknowledged first by the core.

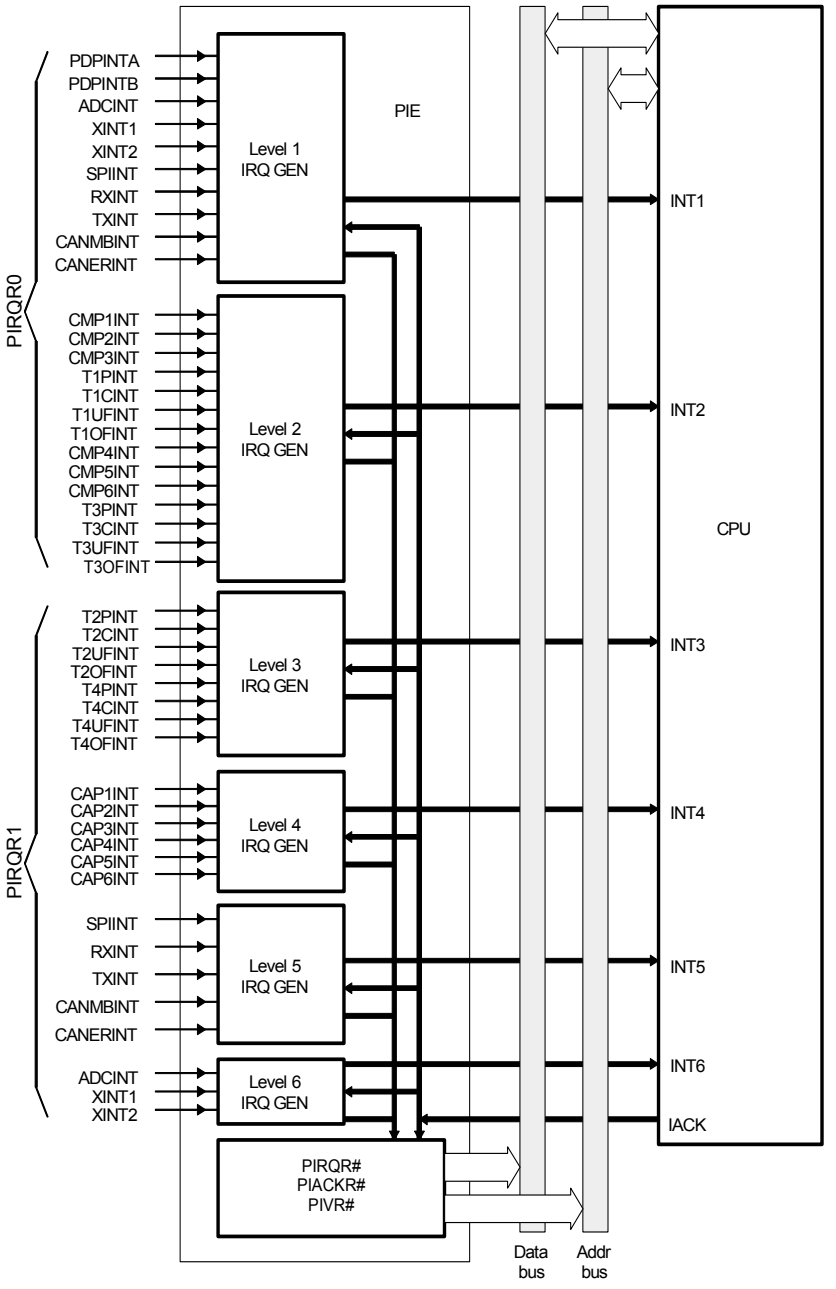

Figure 4.1 Interrupt hierarchy in the LF2407. (Courtesy of Texas Instruments)

4.2.2 Reset and Non-Maskable Interrupts

There are two special interrupts on the LF2407 which have not been covered thus far; the Reset (RS) and the Non-Maskable Interrupt (NMI). Both of these interrupts bypass the usual interrupt hierarchy and feed straight to the DSP core. A reset causes the core to branch to address 0000h in program memory. Resets are activated during power on, when the external RESET pin is brought to logic "0" (O Volts), or by the Watchdog Timer. If the Watchdog is not disabled, it will pull the reset pin to "0" if not periodically reset.

When an illegal memory space is written to, the illegal address flag (ILLADR) in System Control and Status Register 1 (SCSR1) will be set. When this flag is set, a non-maskable interrupt (NMI) will be generated, causing the core to branch to address 0024h in program memory. The illegal address flag (ILLADR) will remain set following an illegal address condition until it is cleared by software or a DSP reset.

4.3 Interrupt Control Registers

This section will review the interrupt control registers. The IFR, IMR, and PIVR registers as well as the INTM bit discussed in the previous section will be presented in more detail. We will not discuss peripheral level interrupt bits in this chapter, as they will be discussed in each section dealing with the specific peripherals.

There are three registers used at the CPU level, the Interrupt Flag Register (IFR), the Interrupt Mask Register (IMR), and the Peripheral Interrupt Vector Register (PIVR). The IFR and IMR control the interrupt signal at the beginning of the CPU level. The PIVR register, while actually loaded in the PIE, provides information about the specific interrupt that occurred at the peripheral level. This information can be used by the ISR in determining the source of the interrupt signal. In addition to these registers, the INTM bit at the CPU level provides the final "gateway" that the interrupt signal must pass through to reach the core itself.

In addition to the peripheral interrupts, there are two External Interrupts (XINT1, XINT2). Their interrupt request operation is exactly like the peripheral interrupts. However, external interrupts are triggered by a logic edge transition on their external pin. The external interrupt control registers will also be discussed.

4.3.1 Interrupt Flag Register (IFR)

The IFR is a 16-bit (only 6 bits are really used) register mapped to address 0006h in data memory. The IFR is used to identify and clear pending interrupts at the CPU level and contains the interrupt flag bits for the maskable interrupt priorities INT1–INT6.

A flag bit in the IFR is set to "1" when an individual interrupt request signal makes its way out of the peripheral level and into the CPU level. The particular flag

bit set depends on what priority the individual interrupt is grouped under. After the interrupt is serviced, the IFR bit corresponding to the interrupt is automatically cleared (to "0") by the DSP.

In addition to triggering the CPU level during the standard interrupt process, the IFR can also be read by software. If a desired situation occurred where the INTM bit was set (meaning no interrupt signals make it to the core) and an interrupt signal was generated at the below levels, the corresponding bit in the IFR would still be set. In this situation, the IFR could be read by software to identify pending interrupt requests.

If desired, to "manually" clear a bit in the IFR, software needs to write a "1" to the appropriate bit (see IFR bit descriptions). The flag bits can be thought of as "toggling" when a "1" is written to them. Loading the IFR into the accumulator, then storing the contents of the IFR back into itself clears all bits in the IFR. However, if the peripheral level interrupt flag bit is still set, the corresponding bit in the IFR will immediately become set right after it is cleared.

Notes:

1. *To clear an IFR bit, we must write a one to it, not a zero.*
2. *When an interrupt is acknowledged,* ***only the IFR bit is cleared automatically****. The flag bit in the corresponding peripheral control register is* ***not*** *automatically cleared. If an application requires that the control register flag be cleared, the bit must be cleared by software.*
3. *IFR registers pertain to interrupts at the CPU level only. All peripherals have their own interrupt mask and flag bits in their respective control/configuration registers.*
4. *When an interrupt is requested by the INTR assembly instruction and the corresponding IFR bit is set, the CPU does not clear the bit automatically. If an application then requires that the IFR bit needs to be cleared, the bit must be cleared by software.*

Interrupt Flag Register (IFR) — Address 0006h

15-6	5	4	3	2	1	0
Reserved	INT6 flag	INT5 flag	INT4 flag	INT3 flag	INT2 flag	INT1 flag
0	RW1C–0	RW1C–0	RW1C–0	RW1C–0	RW1C–0	RW1C–0

Note: *0 = always read as zeros, R = read access, W1C = write 1 to this bit to clear it, -0 = value after reset.*

Bits 15–6 Reserved. These bits are always read as zeros.

Bit 5 INT6. Interrupt 6 flag. This bit is the flag for interrupts connected to interrupt level INT6.

0 No INT6 interrupt is pending

1 At least one INT6 interrupt is pending. Write a 1 to this bit to clear it to 0 and clear the interrupt request

Bit 4 **INT5.** Interrupt 5 flag. This bit is the flag for interrupts connected to interrupt level INT5.
0 No INT5 interrupt is pending
1 At least one INT5 interrupt is pending. Write a 1 to this bit to clear it to 0 and clear the interrupt request

Bit 3 **INT4.** Interrupt 4 flag. This bit is the flag for interrupts connected to interrupt level INT4.
0 No INT4 interrupt is pending
1 At least one INT4 interrupt is pending. Write a 1 to this bit to clear it to 0 and clear the interrupt request

Bit 2 **INT3.** Interrupt 3 flag. This bit is the flag for interrupts connected to interrupt level INT3.
0 No INT3 interrupt is pending
1 At least one INT3 interrupt is pending. Write a 1 to this bit to clear it to 0 and clear the interrupt request

Bit 1 **INT2.** Interrupt 2 flag. This bit is the flag for interrupts connected to interrupt level INT2.
0 No INT2 interrupt is pending
1 At least one INT2 interrupt is pending. Write a 1 to this bit to clear it to 0 and clear the interrupt request

Bit 0 **INT1.** Interrupt 1 flag. This bit is the flag for interrupts connected to interrupt level INT1.
0 No INT1 interrupt is pending
1 At least one INT1 interrupt is pending. Write a 1 to this bit to clear it to 0 and clear the interrupt request

4.3.2 Interrupt Mask Register (IMR)

The Interrupt Mask Register (IMR) is a 16-bit (only 6 bits are used) register located at address 0004h in data memory. It contains a mask bits for each of the six interrupt priority levels INT1–INT6. When an IMR bit is "0", the corresponding interrupt is "masked". When an interrupt is masked, the interrupt will be halted at the CPU level; the core will not be able to receive the interrupt request signal, regardless of the INTM bit status. When the interrupt's IMR bit is set to "1", the interrupt will be acknowledged if the corresponding IFR bit is "1" and the INTM bit is "0". The IMR may also be read to identify which interrupts are masked or unmasked.

Interrupt Mask Register (IMR) — Address 0004h

15-6	5	4	3	2	1	0
Reserved	INT6 mask	INT5 mask	INT4 mask	INT3 mask	INT2 mask	INT1 mask
0	RW	RW	RW	RW	RW	RW

Note: *0 = always read as zeros, R = read access, W = write access, bit values are not affected by a device reset.*

Bits 15–6 **Reserved**. These bits are always read as zeros.

Bit 5 **INT6.** Interrupt 6 mask. This bit masks or unmasks interrupt level INT6.
0 Level INT6 is masked
1 Level INT6 is unmasked

Bit 4 **INT5.** Interrupt 5 mask. This bit masks or unmasks interrupt level INT5.
0 Level INT5 is masked
1 Level INT5 is unmasked

Bit 3 **INT4.** Interrupt 4 mask. This bit masks or unmasks interrupt level INT4.
0 Level INT4 is masked
1 Level INT4 is unmasked

Bit 2 **INT3.** Interrupt 3 mask. This bit masks or unmasks interrupt level INT3.
0 Level INT3 is masked
1 Level INT3 is unmasked

Bit 1 **INT2.** Interrupt 2 mask. This bit masks or unmasks interrupt level INT2.
0 Level INT2 is masked
1 Level INT2 is unmasked

Bit 0 **INT1.** Interrupt 1 mask. This bit masks or unmasks interrupt level INT1.
0 Level INT1 is masked
1 Level INT1 is unmasked

Note: *A device reset does not affect The IMR bits.*

4.3.3 Peripheral Interrupt Vector Register (PIVR)

The Peripheral Interrupt Vector Register (PIVR) is a 16-bit read-only register located at address 701Eh in data memory. Each interrupt has a unique code which is loaded into the PIVR when in the PIE module. When a peripheral interrupt signal is passed through the PIE module, the PIVR is loaded with the vector of the pending interrupt which has the highest priority level. This assures that if two interrupts of

different priorities happen simultaneously, the higher priority interrupt will be serviced first.

Peripheral Interrupt Vector Register (PIVR) — Address 701Eh

15	14	13	12	11	10	9	8
V15	V14	V13	V12	V11	V10	V9	V8
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
7	6	5	4	3	2	1	0
V7	V6	V5	V4	V3	V2	V1	V0
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0

Note: *R = read access, -0 = value after reset.*

Bits 15–0 V15–V0. Interrupt vector. This register contains the peripheral interrupt vector of the most recently acknowledged peripheral interrupt.

External Interrupt Control Registers

The external interrupts (XINT1, XINT2) are controlled by the XINT1CR and XINT2CR control registers, respectively. If these interrupts are enabled in their control registers, an interrupt will be generated when the XINT1 or XINT2 logic transition occurs for at least 12 CPU clock cycles.

For example, if XINT1 was configured for generating an interrupt on a low (0 Volts) to high (3.3 Volts) transition and the XINT1 pin only went high for 6 clock cycles, then back down to low, an interrupt request would not occur. However, if the pin was brought high for 12 or more cycles, an interrupt request signal would be generated.

External Interrupt 1 Control Register (XINT1CR) – Address 7070h

15	14–3	2	1	0
XINT1 flag	Reserved	XINT1 polarity	XINT1 priority	XINT1 enable
RC-0	R-0	RW-0	RW-0	RW-0

Note: *R = read access, W = write access, C = clear by writing a 1, -0 = value after reset.*

Bit 15 XINT1 Flag

This bit indicates if the selected transition has been detected on the XINT1 pin and is set whether or not the interrupt is enabled. This bit is cleared by software writing a 1 (writing a 0 has no effect), or by a device reset.

0 No transition detected

1 Transition detected

Note: *the description in the TI user guide can be misleading: this bit is* ***not*** *cleared automatically during the interrupt acknowledge sequence.*

Bits 14–3 Reserved. Reads return zero; writes have no effect.

Bit 2 XINT1 Polarity

This read/write bit determines if interrupts are generated on the rising edge or the falling edge of a signal on the pin.

0 Interrupt generated on a falling edge (high-to-low transition)

1 Interrupt generated on a rising edge (low-to-high transition)

Bit 1 XINT1 Priority

This read/write bit determines which interrupt priority is requested. The CPU interrupt priority levels corresponding to low and high priority are coded into the peripheral interrupt expansion controller. These priority levels are shown in Table 2–2, *240xA Interrupt Source Priority and Vectors,* in Chapter 2 on page 2-9.

0 High priority

1 Low priority

Bit 0 XINT1 Enable

This read/write bit enables or disables external interrupt XINT1.

0 Disable interrupt

1 Enable interrupt

External Interrupt 2 Control Register (XINT2CR) – Address 7071h

15	14–3	2	1	0
XINT2 flag	Reserved	XINT2 polarity	XINT2 priority	XINT2 enable
RC-0	R-0	RW-0	RW-0	RW-0

Note: *R = read access, W = write access, C = Clear by writing a 1, -0 = value after reset.*

Bit 15 XINT2 Flag

This bit indicates if the selected transition has been detected on the XINT2 pin and is set whether or not the interrupt is enabled. This bit is cleared by software writing a 1 (writing a 0 has no effect), or by a device reset.

0 No transition detected

1 Transition detected

Note: *the description in the TI user guide can be misleading: this bit is* ***not*** *cleared automatically during the interrupt acknowledge sequence.*

Bits 14–3 Reserved. Reads return zero; writes have no effect.

Bit 2 XINT2 Polarity

This read/write bit determines if interrupts are generated on the rising edge or the falling edge of a signal on the pin.

0 Interrupt generated on a falling edge (high-to-low transition)
1 Interrupt generated on a rising edge (low-to-high transition)

Bit 1 XINT2 Priority

This read/write bit determines which interrupt priority is requested. The CPU interrupt priority levels corresponding to low and high priority are coded into the peripheral interrupt expansion controller. These priority levels are shown in Table 2–2, *240xA Interrupt Source Priority and Vectors,* in Chapter 2 on page 2-9.

0 High priority
1 Low priority

Bit 0 XINT2 Enable

This read/write bit enables or disables the external interrupt XINT2.

0 Disable interrupt
1 Enable interrupt

4.4 Initializing and Servicing Interrupts in Software

In order to utilize the interrupt functions of the LF2407, several steps should be taken to initialize the DSP and interrupt related registers. This will assure that no false interrupts are asserted. While it is unlikely that a false interrupt would be generated, writing code that would ignore a false interrupt is good practice.

Servicing the interrupt requires that a few steps also be taken to "reset" the interrupt so that future interrupts of the same origin can also occur.

4.4.1 Configuring the LF2407 for Interrupt Operation

Several steps should be performed via software to prepare the DSP and interrupt system for use before any sort of algorithm is entered. The following provides for a general procedure for initializing the DSP interrupts and peripherals:

1. The first instruction after the START label should be to set the INTM bit. This assures that no interrupts can occur during initialization.
2. Once the INTM bit is set, then the second step is to mask each of the six CPU level interrupts by writing "0" to the IMR.
3. Once all bits in the IMR are "0", the IFR value should be loaded into the accumulator, and then the accumulator should be written to the IFR. This writes the IFR back into itself, thereby clearing all flag bits.

4. Now is the time to disable the Watchdog timer by writing "6Fh" to the WDCR (watchdog control register). Also, the DSP should be configured by setting the System Control Registers (SCSR1,SCSR2) for desired operation and the enabling the clock to desired peripherals.
5. If applicable, set the reset bit in the selected peripheral control registers. Configure peripheral for desired operation.
6. Configure the IMR to unmask only those interrupt levels which correspond to the selected peripheral.
7. Clear the INTM bit to allow future interrupts to reach the CPU.
8. If applicable, bring the selected peripherals out of reset/enable operation via peripheral control register.

Example 4.1 - The following block is a segment of code which provides an example of interrupt initialization.

```
START:
        LDP     #0h             ;set DP=0
        SETC    INTM            ;Disable interrupts
        SPLK    #0000h,IMR      ;Mask all core interrupts
        LACC    IFR             ;Read Interrupt flags
        SACL    IFR             ;Clear all interrupt flags
        LDP     #WDKEY >> 7h    ;Peripheral page
        SPLK    #006Fh, WDCR    ;Disable WD if VCCP=5V
        SPLK    #0000h,  SCSR1 ;
KICK_DOG
        SPLK    #0h,GPR0        ;Set wait state generator for:
        OUT     GPR0,WSGR       ;Program Space, 0-7 wait states
        LDP     #0E1h
        SPLK    #00004h, MCRA   ;Configure XINT pin for primary
        LDP     #0E0h
        SPLK    #5h, XINT1CR    ;Configures XINT1 pin for
                                ;polarity(low to high) priority(high),
                                ;and enable bit
        LDP     # 0h
        SPLK    #1, IMR         ;XINT is INT1 so set IMR to "1"
        CLRC    INTM            ;Enables interrupts to core
LOOP            B   LOOP        ;loops here until interrupt occurs
```

The above code will enable the XINT1 pin to generate an interrupt of INT1 when a "low to high" transition is detected on the pin.

4.4.2 Servicing Interrupts

Each of the interrupt priority levels INT1 through INT6 has a corresponding memory address 0001h through 0006h in program memory to which the core will branch upon receiving the interrupt. The header file *vector.h* assigns the labels "INT1, INT2, …INT6" to addresses 0001h through 0006h. This header file also instructs the core to branch to the corresponding General Interrupt Service Routines (GISR1 through GISR6) labels which are located in the assembly source file.

It is under the appropriate "GISRx" label in the source file where the interrupt service routine (ISR) is written. In the ISR, a variety of algorithms may be used. The ISR is simply an algorithm to which the core will execute whenever it

encounters an interrupt. The first action in the ISR should be to perform a "context save" by saving the value of the accumulator, status registers, and anything else that could change as a result of the ISR, so that when the core exits from the interrupt, it is essentially in the same state as when it entered.

If multiple peripheral interrupts in the same priority level are enabled, then each of these interrupts would cause the core to branch to the same GISRx. In this case, it would be necessary to first run a PIVR reading and selection algorithm under the GISR which would determine what specific interrupt actually occurred. Then the algorithm would then branch to a Specific Interrupt Service Routine (SISR). Example 4.2 is pseudo-code which is an example of the selection algorithm discussed previously.

Example 4.2 – Two peripheral interrupts (RXINT and TXINT) are both assigned to priority level INT1. The following pseudo-code is a sample algorithm to determine which interrupt occurred and service the interrupt.

```
---------------------------------------------------------------------------------------------
GISR1       - GISR1 corresponds to ONLY INT1 interrupts
      Read the PIVR
            Does the PIVR contain the vector for RXINT ?
                  Yes – Branch to R_ISR
                  No – Continue to next instruction
            Does the PIVR contain the vector for TXINT?
                  Yes – Branch to T_ISR
                  No – Branch to ERROR
 R_ISR
This would be the first SISR, the name of the SISR does not matter
User defined algorithm plus reset interrupt for next occurrence and exit ISR

T_ISR
This would be the second SISR, the name of the SISR does not matter
User defined algorithm plus reset interrupt for next occurrence and exit ISR
ERROR
User defined algorithm plus reset interrupt for next occurrence and exit ISR
---------------------------------------------------------------------------------------------
```

Interrupt Vectors

Information on the different peripheral interrupts and their corresponding PIVR codes can be found in Table 4.1, which lists the peripheral interrupt vector codes that load into the PIVR. The vector is essentially an identification number for each interrupt. Note that an interrupt may have a different overall priority and grouping based on the (low or high) priority level that the interrupt is set to in its corresponding peripheral control register. For example**,** XINT1 (high priority) is assigned vector 0001h and is grouped INT1 with overall priority 7. XINT1 (low priority) is still assigned vector 0001h but is grouped INT6 with overall priority 33.

Table 4.1 Interrupt vectors. (Courtesy of Texas Instruments)

Overall Priority	Interrupt Name	CPU Interrupt Vector	Peripheral Interrupt Vector	Maskable?	Source Peripheral	Description
1	Reset	RSN 0000h	N/A	N	RS Pin, Watchdog	Reset from pin, watchdog time out
2	Reserved	- 0026h	N/A	N	CPU	Emulator trap
3	NMI	NMI 0024h	N/A	N	Nonmaskable interrupt	Nonmaskable interrupt

(a) **INT1** *(level 1)*

Overall Priority	Interrupt Name	CPU Interrupt Vector	Peripheral Interrupt Vector	Maskable?	Source Peripheral	Description
4	PDPINTA	INT1 0002h	0020h	Y	EVA	Power drive protection interrupt pin
5	PDPINTB	INT1 0002h	0019h	Y	EVB	Power drive protection interrupt pin
6	ADCINT	INT1 0002h	0004h	Y	ADC	ADC interrupt in high–priority mode
7	XINT1	INT1 0002h	0001h	Y	External interrupt logic	External interrupt pin in high–priority mode
8	XINT2	INT1 0002h	0011h	Y	External interrupt logic	External interrupt pin in high–priority mode
9	SPIINT	INT1 0002h	0005h	Y	SPI	SPI interrupt in high–priority mode
10	RXINT	INT1 0002h	0006h	Y	SCI	SCI receiver interrupt in high–priority mode
11	TXINT	INT1 0002h	0007h	Y	SCI	SCI transmitter interrupt in high–priority mode
12	CANMBINT	INT1 0002h	0040h	Y	CAN	CAN mailbox interrupt (high–priority mode)
13	CANERINT	INT1 0002h	0041h	Y	CAN	CAN error interrupt (high–priority mode)

(b) **INT2** *(level 2)*

Overall Priority	Interrupt Name	CPU Interrupt Vector	Peripheral Interrupt Vector	Maskable?	Source Peripheral	Description
14	CMP1INT	INT2 0004h	0021h	Y	EVA	Compare 1 interrupt
15	CMP2INT	INT2 0004h	0022h	Y	EVA	Compare 2 interrupt
16	CMP3INT	INT2 0004h	0023h	Y	EVA	Compare 3 interrupt
17	T1PINT	INT2 0004h	0027h	Y	EVA	Timer 1 period interrupt
18	T1CINT	INT2 0004h	0028h	Y	EVA	Timer 1 compare interrupt
19	T1UFINT	INT2 0004h	0029h	Y	EVA	Timer 1 underflow interrupt
20	T1OFINT	INT2 0004h	002Ah	Y	EVA	Timer 1 overflow interrupt
21	CMP4INT	INT2 0004h	0024h	Y	EVB	Compare 4 interrupt
22	CMP5INT	INT2 0004h	0025h	Y	EVB	Compare 5 interrupt
23	CMP6INT	INT2 0004h	0026h	Y	EVB	Compare 6 interrupt
24	T3PINT	INT2 0004h	002Fh	Y	EVB	Timer 3 period interrupt
25	T3CINT	INT2 0004h	0030h	Y	EVB	Timer 3 compare interrupt
26	T3UFINT	INT2 0004h	0031h	Y	EVB	Timer 3 underflow interrupt
27	T3OFINT	INT2 0004h	0032h	Y	EVB	Timer 3 overflow interrupt

(c) **INT3** *(level 3)*

Overall Priority	Interrupt Name	CPU Interrupt Vector	Peripheral Interrupt Vector	Maskable?	Source Peripheral	Description
28	T2PINT	INT3 0006h	002Bh	Y	EVA	Timer 2 period interrupt
29	T2CINT	INT3 0006h	002Ch	Y	EVA	Timer 2 compare interrupt
30	T2UFINT	INT3 0006h	002Dh	Y	EVA	Timer 2 underflow interrupt
31	T2OFINT	INT3 0006h	002Eh	Y	EVA	Timer 2 overflow interrupt
32	T4PINT	INT3 0006h	0039h	Y	EVB	Timer 4 period interrupt
33	T4CINT	INT3 0006h	003Ah	Y	EVB	Timer 4 compare interrupt
34	T4UFINT	INT3 0006h	003Bh	Y	EVB	Timer 4 undeflow interrupt
35	T4OFINT	INT3 0006h	003Ch	Y	EVB	Timer 4 overflow interrupt

(d) **INT4** *(level 4)*

Overall Priority	Interrupt Name	CPU Interrupt Vector	Peripheral Interrupt Vector	Maskable?	Source Peripheral	Description
36	CAP1INT	INT4 0008h	0033h	Y	EVA	Capture 1 interrupt
37	CAP2INT	INT4 0008h	0034h	Y	EVA	Capture 2 interrupt
38	CAP3INT	INT4 0008h	0035h	Y	EVA	Capture 3 interrupt
39	CAP4INT	INT4 0008h	0036h	Y	EVB	Capture 4 interrupt
40	CAP5INT	INT4 0008h	0037h	Y	EVB	Capture 5 interrupt
41	CAP6INT	INT4 0008h	0038h	Y	EVB	Capture 6 interrupt

(e) **INT5** *(level 5)*

Overall Priority	Interrupt Name	CPU Interrupt Vector	Peripheral Interrupt Vector	Maskable?	Source Peripheral	Description
42	SPIINT	INT5 000Ah	0005h	Y	SPI	SPI interrupt (low priority)
43	RXINT	INT5 000Ah	0006h	Y	SCI	SCI receiver interrupt (low–priority mode)
44	TXINT	INT5 000Ah	0007h	Y	SCI	SCI transmitter interrupt (low–priority mode)
45	CANMBINT	INT5 000Ah	0040h	Y	CAN	CAN mailbox interrupt (low–priority mode)
46	CANERINT	INT5 000Ah	0041h	Y	CAN	CAN error interrupt (low–priority mode)

(f) **INT6** *(level 6)*

Overall Priority	Interrupt Name	CPU Interrupt Vector	Peripheral Interrupt Vector	Maskable?	Source Peripheral	Description
47	ADCINT	INT6 000Ch	0004h	Y	ADC	ADC interrupt (low priority)
48	XINT1	INT6 000Ch	0001h	Y	External interrupt logic	External interrupt pins (low–priority mode)
49	XINT2	INT6 000Ch	0011h	Y	External interrupt logic	External interrupt pins (low–priority mode)
	Reserved	000Eh	N/A	Y	CPU	Analysis interrupt
N/A	TRAP	0022h	N/A	N/A	CPU	TRAP instruction
N/A	Phantom Interrupt Vector	N/A	0000h	N/A	CPU	Phantom interrupt vector

4.5 Interrupt Usage Exercise

This exercise will help the reader become familiar with interrupt operation and writing interrupt service routines in software. The skills practiced in this exercise are extremely relevant in sequential chapters where interrupts must be understood for peripheral use.

1. Create a new project and source file each named "lab4". Add the same header files as in previous exercises.
2. Create a program which first properly configures the LF2407 and XINT1 interrupt registers for operation on a low to high clock edge. On the LF2407 EVM module, jumper the XF pin the XINT1. Configure the XF pin to initially output logic "0". The program should utilize a looping algorithm and the XINT1 interrupt to perform the following tasks:

a. Start with the value "0h" in the accumulator. Store the number in the accumulator to the data memory address 300h. When this operation is complete, set the XF pin to be "1" (logic high). This should trigger an XINT1 interrupt.
b. In the ISR, keep count of the number of interrupts generated in the address "030Fh". Start counting at "0" for the first interrupt generated. Reset the XF pin to logic "0". Re-enable the interrupt.
c. Keep repeating steps (a) and (b), but use the numbers "0001h" through "000Ah" instead and store them to memory address 301h through 30Ah.

The program should store a total of 11 numbers (0h to Ah) to memory addresses 300h through 30Ah. There should be exactly 11 interrupts counted with the number "Ah" stored in memory address 30Fh.

d. After steps (a) through (c) are complete, perform the calculations "Ah" multiplied by "3h" , "1h" multiplied by "5h", and "11h" multiplied by "7h" one after the other. Create a transition on the XF pin so an XINT1 interrupt will be generated after each calculation is complete. Count the number of these interrupts and store them in data memory address 310h.

When the program is finished with the task, have it loop infinitely until halted by the user. This exercise is now concluded.